Shahar Frank, CTO & Founder

*Intro
Founded in 2013
GA - v1 Q4 2016

Elastic Cloud File System (ECFS)
Scalable data platform
- 1000s of nodes, 100000s of clients
- 100s thousands FS (data containers), unlimited files/directories
- Exabyte scale capacity, 10s millions IOPS (and above)
- advanced embedded data management and analytics

Software only, virtualisation and cloud friendly
- runs on physical servers (tested on 20+ platforms)
- runs on on-prem virtualisation (VMware, KVM, etc)
- runs on cloud VMs (Amazon, Google, etc)

Flash Native
- supports all NAND technologies interfaces (flash tiering, SCM ready)
- supports object/S3 cold tier (dynamic tiering/HSM/ILM)

Architectural Coverage
Enterprise Level Features
- NDU, n-way redundancy, self healing, snapshots, sync/async DR
Storage Interfaces
- NFSv3/v4, SMB2/3, S3, HDFS
Cloud features
- multi-tenancy, QoS, Hot add/remove nodes/capacity
- snapshot shipping to S3 (“CloudConnect”)
- ILM/HSM/Dynamic tiering to other sites/clouds, Object/S3
- multi-site (geographically distributed access)
Deployment modes
- hyperconverged mode (HCI)
- Dedicated storage mode (DSM) - with 2 tiers or single tier
- cloud (“Marketplace”) - as a service / as an application

Use Cases
Legacy Enterprise
- virtualisation back-end storage
- mixed workload consolidation
- general NAS file sharing
- “SDS”
Cloud and Next Gen
- Lift and shift to cloud
- DR to cloud
- Stateful containers
High Performance
- Life Sciences
- Financial Services

- EDA
- Oil & Gas
Cross-cloud
- cloud-integrated analytics
- geo-distribution across clouds

*Design Objectives and Journey

A successful architecture
“A good trade-off between all relevant dimensions, resources and requirements yo produce the best solution for the desired target(s)”
It’s not a linear path

“Be the best data platform for the cloud era enterprise”

Elastifile Architectural Base Observations
1. Enterprises increasingly use clouds or cloud-like techniques
2. In a cloud (like) environment the focus ISN’T infrastructure (storage) but rather services (data)
3. Data services must be implemented by simple, efficient mechanisms for many concurrent I/O data patterns
4. Everything should be managed by APIs
5. Data management should be very fine grained
6. Data mobility has to be solved

Elastifile Architectural Base Requirements
- Everything must be automatic
- Avoid unnecessary restrictions and limitations. Assume as little as you can about the customer’s IO patterns
- BYOH: Avoid unnecessary/uncommon hardware requirements like NVRAM, RDMA networks, etc (optional optimisations OK)
- support realtime & dynamic reconfiguration of the system
- support heterogeneous hardware (CPU, memory, SSD types, sizes, etc)
- provide good consistent predictable performance for the given resources (even under failures and noisy environments)

Elastifile Architectural Base Decisions
Scale-out (and Scale-up)
- cloud, cloud and cloud!
Software only
- cloud and virtualisation friendly, cost effective
Flash only
- provides flexibility and efficient multiple concurrent IO patterns
- capacity efficiency achieved by dedupe, compression and tiering
Application level file system
- enables unique data level services and the best performance (!)
- superset of block/object interfaces
- enables data sharing (and not only storage sharing) for user self-service.

Bumps in the Road

Private clouds didn’t happen as expected
- OpenStack didn’t gather enough momentum
- it seems that private clouds don’t make sense short of the web-scale guys - lack of economy of scale
- many enterprises do not attempt to modernise their legacy systems, but rather attempt to shift (some/all) workloads to the public cloud
Impact? Had to support public cloud use cases earlier than expected (a good thing in the end)

HCI model is problematic for many use cases
- not perceived well by many storage admins due to problematic responsibilities boundaries (“where is my storage?)
- requires coordination with the apps/server infrastructure
- limits the ability to scale resources (e.g. scale capacity, not performance)
Still HCI is a good fit for
- Places/use cases without (proper/skilled) IT (e.g. ROBO, SMB)
- Vertical implementations (web scale companies in most places)
Impact? Added dedicated storage mode to provide separate storage resources and scaling

*Ezra Hoch, Chief Architect
Architectural Foundations
- Data Path
- metadata
- clustering

Goals and Motivation
Cloud-ready (on-prem / public)
- scalable data and metadata
- work well in noisy and fluctuating environments
- heterogeneous hardware
- access patterns vary quickly
Enterprise grade file system
- POSIX semantics
- advanced data services (snapshots, async DR, etc)
Consistent high-performance & low latency
- predictable data path
- no data-caching

Internal Data Representation
Internally, we manage objects
- data and metadata are persisted as variable-length objects
- these objects are used to construct our file / dir structure
Objects can be owned by different nodes
- scale-out by dynamic fine-grained ownership of objects

[File Encoding / Layout]

[Dir Encoding / Layout]

[Main Data Path Components]

Redundancy and Data Path Model
- n-way Replication
- persistency before ack-ing the user (no write cache or NVRAM) - complete power-failure resistant
- distributed write-anywhere semantics
- metadata path is completely separated from data path
- consistency is preserved by a distributed transaction layer - metadata is journaled anywhere in the cluster, persisted as a distributed journal (not a fixed location)
- failure detection doesn't rely on special hardware - failure handling doesn't rely on power or network fencing

Scalable Metadata
In many case, the bottleneck is the metadata
For example
- Small files
- creates / deletes
Common solution: siloing (customers don't like)
scaling metadata is hard, due to the consistency requirements of file-systems
usually, achieving consistency comes at the cost of performance (aka CAP Theorem)
We're using "consistent layering" to get a good combination of consistency and performance at scale

[Consistency layering - Elastifile_ConsistencyLayering.jpg]

Bizur (distributed or decentralised in Hebrew): A Key-value Consensus Algorithm
Specialized consensus algorithm (instead of Paxos / Raft)
Optimized for high concurrency and low latency, especially during failures
Achieves the above by utilising a (simplified) key-value data model
- provides consistency at the key level vs the entire dates level
- provides efficient cluster-wide operations (e.g. CAS)
Published a paper about it - https://arxiv.org/pdf/1702.04242.pdf

Clustering: Overview
- Configuration DB: persistent, HA, fault-tolerant, consensus
- Nodes monitor each other continuously and report errors
- Clustering officer: elected via ECDB, datapath-rate clustering decisions
- Clustering service: control-rate clustering decisions

Clustering: Failure Handling
- Node fails
- New CO is elected
- Fencing of failed node
- IOs can continue (within ~1 second from failure)
- ECS rebuilds ROC, ORC and ECDB

Clustering: Node Replacement
- A new node can be added to the system and replace the failed node

Subscription-based, capacity licensing

*Demo on Node failure handling / downsize, dedicated storage mode (VMware)